Fundamental Semantics of Data Modeling

- formerly -

YADMD (Yet Another Data Modeling Document)

Keri Anderson Healy

rev. December 1, 1989

Preface

This document was originally written to serve as the requirements document for a graduate-level student project. Since that original purpose, the document has found its way into the hands of several different audiences, each with an interest in (and a variety of experience levels with) data modeling semantics. Very useful feedback has given and incorporated into revisions of this document -- to the point where the contents now reflect a blending of the input of several individuals. In fact, the editing and enhancement of the document moved into the hands of a small "brown bag" group of data bigots in the Pacific Northwest who chew over its contents in between bites of tofu & cheese sandwiches. The latest enhancement has been the addition of examples taken from my *Data Architecture* course -- showing we do listen to our feedback, in this case a cry for 'relief' from the abstract treatment of many of the topics.

If you have comments, kudos, additions, or corrections, we would like to hear from you. If you have received this copy indirectly and would like to receive any updates as they emerge, also let us hear from you (please tell us what revision you have). It will help us focus the future contents if you let us know something about how you've used the document and your experience level with data modeling.

Keri & the Automated Reasoning Corporation
721 Park Ave., Bainbridge Island, WA, USA 98110

Table of Contents

1. INTRODUCTION

With the growing popularity of "CASE" (Computer Aided Software Engineering), modeling tools are all the rage. Yet, is there something more to a data model than the "pretty face" now allowed by personal workstations with high-resolution graphics, oodles of memory, and many MIPS? Each modeling tool seems to depict the model in a slightly different fashion. Is this difference merely cosmetic, or does it signify some fundamental difference in the underlying technique?

This document describes the requirements for a data modeling tool by defining the fundamental semantics which any industrial-grade data modeling tool should support. Its approach will be to define formally a set of "data modeling primitives" common to the data modeling discipline, from which technique- (and product-) specific constructs may be derived. The concepts will be illustrated by reference to two popular data modeling techniques, the Chen ER (Entity Relationship) model [CHEN76],[FLAV81] and the Data Modeling Technique (DMT) [LOOM87], [BROW82], [BROW83].

John Zachman's A Framework for Architectures [ZACH84] will be used to describe the context in which these two data modeling techniques are applied. For those readers familiar with the vocabulary of Zachman "cells", Chen's ER/Information Modeling (ER/IM) constructs are used for the intersection of the "User row" and the "Data column" *(Business-perspective Information Model)* and the DMT constructs for the "I/S row" and the "Data column" *(I/S-perspective Conceptual Data Model)* [ref. FIG. 1-2]. Some would argue that one technique (or the other) can/should be applied to both cells – that a uniform representation is less confusing and makes cell transition easier. The choice to distinguish the techniques (at least for the discussion in this paper) was made for several reasons:

- In a teaching environment, a survey of data modeling approaches can be presented, highlighting their similarities and distinctions more readily using two graphical representations; there is visual reinforcement for the distinctions in concept.

- In an applied environment, the use of distinct notations follows the precept of orthagonality – things that are different should look distinctively different. Business perspective information modeling does not model the same thing that the I/S analyst captures in a data model. Unfortunately, this distinction is not always apparent. (This is the topic of another paper, at another time.)

- The original Chen ER notation as applied in Information Modeling is a simple graphic 'language' that can be readily explained to an end user (business person). Extending this simple notation with a myriad of special symbols forces it to lose some of its original charm, namely its unintimidating simplicity for a non-technical user.

- The DMT notation uses more symbol types, allowing the modelers to assert richer semantics graphically. But this added complexity also marks it clearly as a tool for the rigorously trained analyst. While the notation is still fairly simple, the subtleties of the symbols and their combinations cannot be visually validated by a casual user. The DMT notation serves better as a form of 'analytical shorthand' (analyst-to-analyst

communication) rather than as a direct communication tool between business user and analyst.

My preference for applying these two techniques in this manner is entirely personal. Other data modelers elect to apply these (or other) techniques at different points in the development life cycle. I have expressed this preference mainly to provide examples; data modeling automation support would not need to implement such biases.

For completeness, this paper refers to a third perspective of data model, the *Technology* (or DBMS) perspective data model, also often labeled the "physical model" [ref. FIG. 1-2]. This is the technology-constrained data model, targeted for a relational, network, or hierarchical implementation in a specific DBMS (e.g., DB2, IDMS, IMS, etc.). In this paper only the relational model will be used to represent the Technology-perspective model. Variations for the other DBMS models are well documented in the literature [DATE86],[LOOM87].

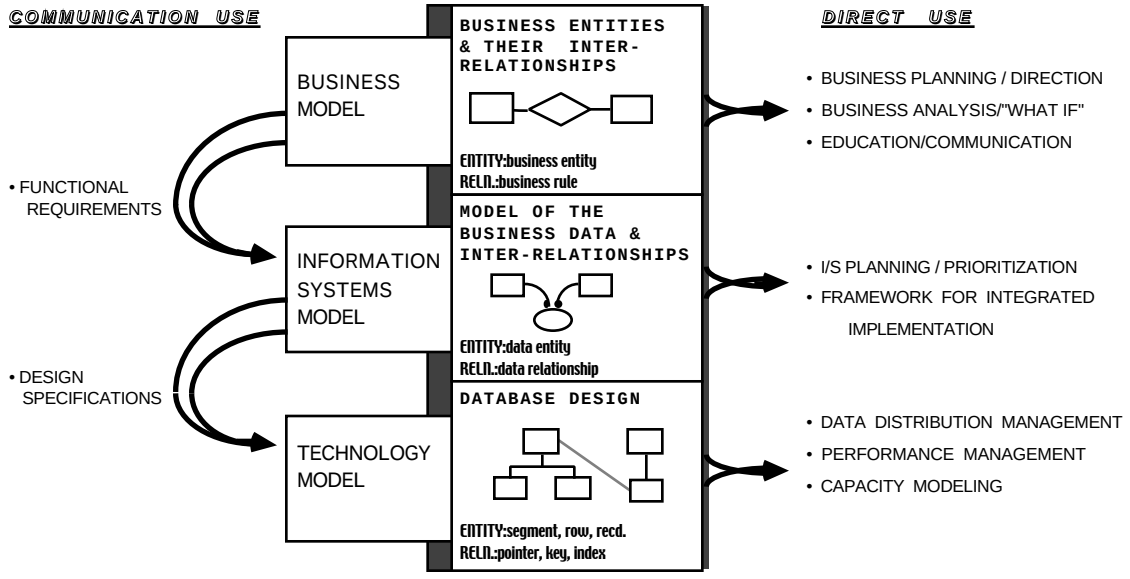
At each perspective in the Zachman framework, a data model can support two kinds of use:

(1) direct use – where the information captured by the data model definition directly supports some use at its own perspective, and

(2) communication use – where the data model can play a role in communicating from one perspective to another. [ref. FIG. 1-1]

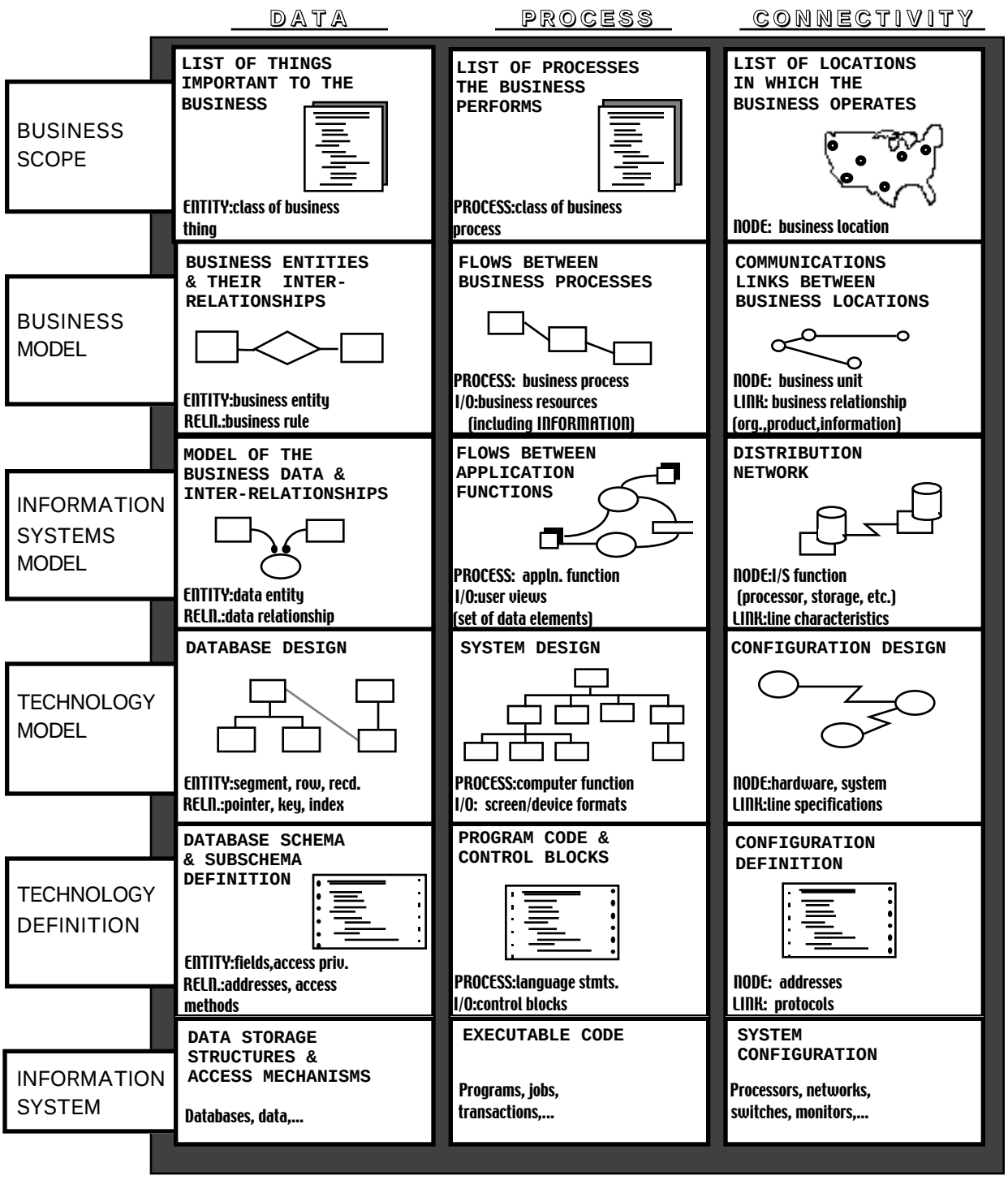
In this latter case, there are heuristics and algorithms which can be applied to a data model to transform it from a *higher* to a *lower* perspective model (i.e., from *BUSINESS* to *CONCEPTUAL*, from *CONCEPTUAL* to *TECHNOLOGY*). While this is the focus of other work in progress, the documentation of these transformation rules is beyond the scope of this paper.

FIG. 1-1: APPLICATIONS OF THE DATA MODEL



This document does not attempt to instruct in any particular data modeling method (i.e., to tell the reader 'how to' develop a model, nor does it delve into the issues of model content validation and integrity). Such topics have been well-documented by others. It is assumed that the reader knows why the model is being developed and how to do it. In other words, the reader is assumed to be a *data bigot* already – or know where to find one....

FIG. 1-2: ZACHMAN'S ARCHITECTURE FRAMEWORK



GUIDE/Information Systems Architecture Group based on work by John A. Zachman, IBM

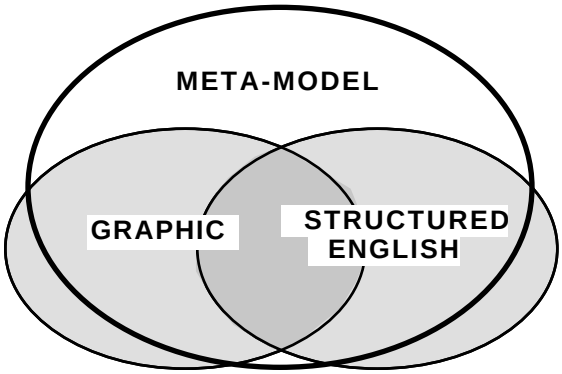
2. MODEL REPRESENTATIONS

A data model can take several forms. Since the most familiar (and most debated) representation form is graphic, that form is discussed first. Structured English[1] and syntax-based meta-model declaration are two other representation types described. Translations of the model from one representation form to another will be discussed in later sections.

In a modeling tool, each of these three representation forms should support both input and output functions. For example, the data analyst might enter the data model definition graphically and produce a Structured English output report for review with the user. Alternatively, the meta-model definition syntax might be loaded "in batch" from some other automated source (e.g., a host-based data repository) and a graphic model produced as an output.

Both the graphic and the Structured English representations have inherent limitations on the type and amount of information they can depict easily. Thus, they should be viewed as semantic subsets of the meta-model definition, which stands as the complete and rigorous data specification. In turn, both the graphic and Structured English representations may have their own "process control" attributes which are unique to their representation form (e.g., shape placement attributes for the graphic representation).

FIGURE 2-1: MODEL REPRESENTATIONS



The meta-model represents the declarative data definition of all the business properties captured in the data model. Typically it serves as the stored representations ("database") from which other representations (graphic & Structured English) are constructed.

[1] "English" in this context means any national language. The examples are presented in English, but it should be accepted that suitable translations can be developed in other languages.

2.1 GRAPHIC REPRESENTATION

2.1.1 BASIC SYMBOL SET

Fundamental data modeling concepts can be represented graphically as three symbol types:

- a **shape** (or node),
- a **connector** (or arc) which joins two[2] shapes, and
- **text**, which labels a shape or connector.

These symbols form a set of graphic primitives which can be assembled to support a variety of data modeling techniques. Depending on the approach, each symbol may take on different meaning. There can also be technique-dependent constraints for assembling symbols into a completed model. In this section, the symbol descriptions are presented *out of context*; a variety of *in context* illustrations will be given later with specific modeling topics.

An interactive graphic tool to support data modeling can enforce some of the specific assembly and connection rules and thereby support a specific method (or set of methods). Alternatively, it can remain a general graphic tool and only adopt certain basic modeling behaviors. The following graphic behaviors should be considered a useful, mandatory minimum:

- A shape may be moved about freely on the screen. When moved, its connectors should move with it. The attachment points of a connector may need to be adjusted automatically as a shape is moved.

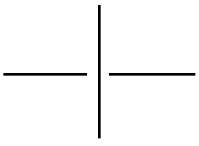
- A shape's labels move with it.

- A connector's labels move with it.

- The connector should automatically place itself for the optimal connection path between two shapes. However, the user may later move the connector to achieve a more aesthetic connection path.

- Multiple connectors between the same two shapes must be drawn unambiguously.

- Connection lines should be drawn to minimize line crossings. However, when it cannot be avoided, an unobtrusive but non-ambiguous line-crossing representation should be used, e.g.,



[2] in some special cases, a connector can join one shape to itself.

- When a diagram is saved, it must be able to 'remember' the latest placement designated by the user. In other words, while the placement may be developed algorithmically from the contents it may also reflect an arbitrary placement designation by the modeler.

- Self-to-self connectors (an arc with the same node at both ends) may be drawn; multiple self-to-self connectors are possible (although there may be a physical practical limit to the number which may be depicted graphically).

Some tools allow further refinement of the placement by the user:

- explicitly designate (and fix) the point where a connector attaches to a shape.

- relocate (and fix) the placement of a label on a shape or connector.

- group shapes into more complex shapes (either temporarily, e.g., while performing an edit function, or as a permanent aggregate object).

In the Macintosh environment, the product Design is a good example of a general (methodology-neutral) graphic tool which exhibits most of the behavior characteristics listed above.

2.1.1.1 SHAPES

BOX

COMMENTS:
- the basic rectangle, 'wide' rather than 'tall'

CONNECTOR RULES:
- can attach at any point on any side

LABEL PLACEMENT:
- inside / centered
- outside / top / aligned with left side

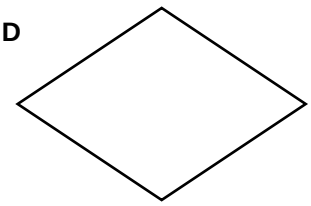
PUFFY BOX

COMMENTS:
- rectangle with rounded corners, 'wide' rather than 'tall'

CONNECTOR RULES:
- same as for BOX

LABEL PLACEMENT:
- same as for BOX

DIAMOND



COMMENTS:
- 'wider' than 'tall'

CONNECTOR RULES:
- can only attach at the points

LABEL PLACEMENT:
- inside / centered

2.1.1.2 CONNECTORS

CONNECTOR GRAPHICS

LINE

DASHED LINE

DOUBLE LINE

Note: these connector lines may also include "bent" versions of a continuous line. The bend in the line may either be at a right angle or use a rounded bend depending on context. A connector line may have more than one bend.

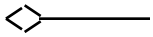
CONNECTOR ATTACHMENTS

Connectors may optionally have additional symbol(s) at either (or both) end(s) where the connector intersects the shape.

BIG DOT

DOT is solid black fill

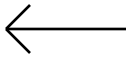
SMALL DIAMOND



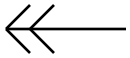
DIAMOND is solid white fill

The following are not used in ER/IM or DMT but are useful graphic shapes found in other modeling approaches:

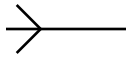
ARROWHEAD



DOUBLE ARROWHEAD



CROW'S FOOT



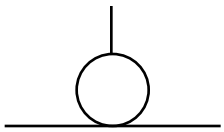
2.1.1.3 LABELS

Labels are text strings associated with either a shape or a connector. The placement of the label varies with the shape and with the specific modeling approach. Text can include both character and numbers. There may be more than one label associated with a given symbol.

2.1.2 EXTENDED SYMBOL SET

2.1.2.1 EXTENSIONS TO SHAPES

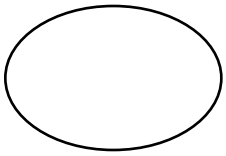
SMALL CIRCLE



CONNECTOR RULES (shown in example):
- one is tangent
- one is perpendicular to an imaginary tangent
- connector line is always solid & single

LABEL PLACEMENT:
- begins in the circle; extends to the right

ELLIPSE



COMMENTS:
- 'wide' rather than 'tall'

CONNECTOR RULES:
- can attach at any point

LABEL PLACEMENT:
- inside / centered

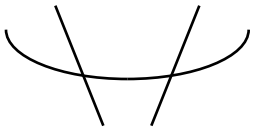
2.1.2.2 EXTENSIONS TO CONNECTORS

HATCHED LINE

"GROWS OUT OF" LINE

SET SYMBOL

CONNECTOR ARC SYMBOL



2.1.2.3 EXTENSIONS TO LABELS

In some cases, other special text symbols (e.g., “<“ or “>“) can be part of the label text.

2.2 STRUCTURED ENGLISH

Structured English is a stylized way of structuring the information content of the data model into a sentence format which can be easily read by a non-technical user. It is typically used as an output representation, either as hardcopy documentation or as part of an interactive requirements dialog, asking for validation or probing to flesh out incomplete information. The grammar of structured English is very precise; when used as an input form, parsing free-form natural language text to deduce requirements is not required.

2.3 META-MODEL DEFINITION

The meta-model is the declared data definition of all the business properties captured in the data model. Typically, it serves as the stored representation from which the graphic and structured English representations are constructed. In this paper, a form of BNF is used for the meta-model representation.

(Refer to Appendix A for an explanation of the BNF notation used in the meta-model examples.)

3. ENTITIES

3.1 CONCEPTS

An entity is *"a person, place, thing, concept [or event] of interest to the business."* At the I/S perspective, this definition is further qualified as *"...and about which the business is interested in storing data."* At the Technology perspective, an entity defines the structures which contain that data.

An entity is defined by business-backed policy points. The definition of entities, as well as other model components, depends on discovering the *rules*, *laws*, and *conventions* of the business being modeled. These rules and laws are considered to be the *business policy.* Often this policy is not neatly written down in one place, and much of data analysis focuses on researching and reconciling this policy over time. [FLAV81] The graphic model reflects in a symbolic form much of this business policy, with each graphic construct representing a specific policy point; thus rigorous constraints are placed on the use of certain symbols and on the allowable symbol combinations in a model. The structured English representation is also an effective way to present the model as a specification of known policy for the business. The meta-model captures free-form text narrative from the policy sources, and it is annotated by source, with a *current confidence rating* given that source. This narrative is useful during data analysis and is used in the direct production of documentation publications generated by the modeling tool.

3.2 REPRESENTATION OF ENTITIES

3.2.1 STRUCTURED ENGLISH

An entity is expressed as a noun, either as the subject or object of a simple sentence. The Structured English grammar for an entity expresses all the relationships in which the entity participates. It also states the business properties *(attributes)* of the entity meaningful to the business scope. Syntax examples can be found in the relationship documentation in 4.1.2.1, 4.2.2.1, 4.3.2.1, 4.4.2.1, 4.5.2.1 and in the attribute documentation in 5.1.2.1. (ref. also [CHEN83])

3.2.2 GRAPHIC ALTERNATIVES

3.2.2.1 BUSINESS-PERSPECTIVE MODEL

An entity is represented as a BOX shape, with the entity name label inside the box. The entity name label is shown in upper case. The entity name is always singular.

3.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL

An entity is represented as either a BOX or a PUFFY BOX shape, with the entity name label outside the box, aligned with the upper left corner. The entity name label is shown in upper case.

Independent Entity
- can exist independently
- has its own identity

Also referred to as:
- KERNEL entity
- STRONG entity

Dependent Entity
- cannot exist independently
- depends on another for part of its identity

Also referred to as:
- WEAK entity

Attributes (domains, possibly role named) may be shown inside the box. The discussion of attributes in the I/S-perspective model is deferred until Section 5.

3.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

By the time the data analyst reaches the technology perspective, there's a real urge to be looking at "token" along with "type." Thus, at the technology perspective an entity is typically represented as an *instance table* showing the attributes of the table with example values. An instance table representation is drawn as a BOX with the table (entity) name label outside the box, aligned with the upper left corner. The table name label is shown in upper case. Attributes (domains, possibly role named) appear inside the box as column headings; rows, if shown, are tuples with example instance values.

CUSTOMER

ID	NAME	CITY	REP
C1	James	Seattle	R23
C2	Kent	Bellevue	R88
C4	Cushing	Olympia	R23
C9	Nelson	Kent	R01
C11	Hays	West Seattle	R20
C21	Miles	Seattle	R20
C70	Jones	Winslow	R01

Another popular representation of the relational model is the declarative function-like syntax popularized by C.J. Date in the pre-DB2 days. [DATE81]

RELATION CUSTOMER (ID,NAME,CITY,REP)

In this example, CUSTOMER is the entity/table name and the labels listed inside the brackets are the attribute names. Unlike the instance table form, this representation does not depict example values.

3.2.3 META-MODEL TO REPRESENT ENTITIES

ENTITY::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<seq[?]>;<id_reference>;<text>
* [**CO**MMENT::=<seq[?]>; <id_reference>;<text>]

* Multiples allowed

4. RELATIONSHIPS

Entities in a data model are linked by relationships. The notion of "relationship" is a complex one – evoking heated, often emotional, debates between data modelers! A symptom of this schism is the reality that this singular concept can be represented either by a SHAPE symbol or as a CONNECTOR symbol joining two SHAPE symbols. This is not merely a graphic distinction but a reflection that this concept is "context sensitive" to the model type – an assertion which will be explored in this section.

In this discussion of relationships, five fundamental types of relationships will be examined:

- **ASSOCIATION RELATIONSHIP**
- **GENERALIZATION RELATIONSHIP**
- **ROLE-PLAYING RELATIONSHIP**
- **AGGREGATION RELATIONSHIP**
- **CHARACTERISTIC RELATIONSHIP**

A sixth type of relationship, a **CLASSIFICATION RELATIONSHIP**, associates an entity instance with a generic type. This is referred to as a TOKEN-TYPE relationship and is generally not of concern in classic data modeling disciplines except as used in example 'instance tables' or in bottom-up techniques of discovery. [TSIC82] However, it is interesting to note that this concept does come into play in some Artificial Intelligence/Expert System (AI/ES) applications where intelligence at the level of instance may be encoded in the knowledge representation. This relationship type is also applicable in Object-Oriented (OO) approaches. Classification Relationships will be discussed superficially in the topic on Generalization Relationships but will not be examined in-depth.

The five relationship types which will be discussed in this section share common properties:

- *RELATIONSHIP NAME*
- ***CARDINALITY*** – for any entity pair in the relationship, how many instances of one entity type can be related to the other entity type?
- ***PARTICIPATION*** – can an instance of one entity in the relationship exist without an instance of the other(s)?

Although all relationship types share these properties in common, each has specific constraints on the values these properties can assume. Relationship types also imply facts about the types of entities which are related. Finally, relationship types restrict the kind of set operators which can be applied to the data model. (ref. [KAMP86] and [GILM87])

The cardinality of a relationship is defined for each entity in the relationship. Typical cardinality values are:

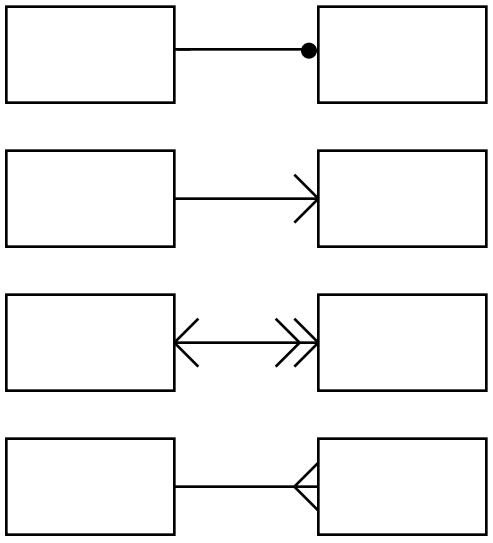
1:1	(One-to-One)	For each ***A*** there is one ***B***; for each ***B*** there is one ***A.***
1:M	(One-to-Many)	For each ***A*** there are many ***B***s; for each ***B*** there is one ***A***.
M:1	(Many-to-One)	For each ***A*** there is one ***B***; for each ***B*** there are many ***A***s.
M:M or M:N	(Many-to-Many)	For each ***A*** there are many ***B***s; for each ***B*** there are many ***A***s.

Cardinality expresses the upper limit constraint of a relationship. Some techniques also allow a lower limit to be expressed explicitly, but generally "1" is the assumed lower limit. For example,

For each ***A*** there are [from 1 to] many ***B***s;

Cardinality can also be represented graphically by some symbol (e.g., arrowhead, big dot, crow's foot) at the end of the connector line. Each symbol type has an associated cardinality value, e.g., 1, M[any], or some specific value. However, the meanings of the various connector symbols can vary with each modeling approach. For example, some techniques use a single arrowhead to represent "1" and a double arrowhead to represent "M"; other approaches let the single arrowhead stand for "M" and no connector symbol for "1".

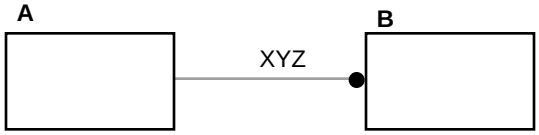
FIG. 4-1: SOME ALTERNATIVES FOR 1:M REPRESENTATION



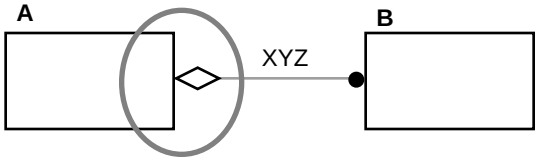
If the graphic portion of the modeling tool is to be "smart" i.e., capture the **meaning** of the symbol and use that meaning in subsequent validation and completeness checking, there needs to be a profile function which enables the analyst to explicitly designate the symbols used

and the concept which each symbol represents. On the other hand, if the tool is merely a graphic drawing tool (like MacDraw or MacFlow), no such functionality need be developed.

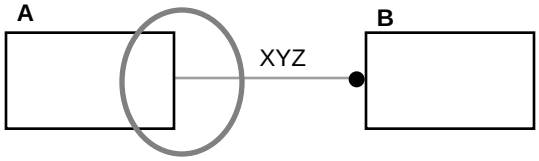
Participation is different from cardinality. Early modeling techniques often blurred this distinction by combining the notions of cardinality and participation, e.g., Many -to-1 or Many-to-Zero. More recent approaches now break participation out as a distinct concept which allows special cases (such as "optional, 2-12") to be unambiguously defined. The concept of participation can be illustrated in the following simple binary Many-to-1 association case, using DMT constructs.



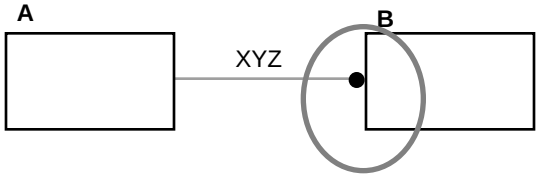
1. Can there be a *B* which does not participate in an *XYZ* relationship with some *A*?
   If yes, the A:B relationship is *"onto"*...



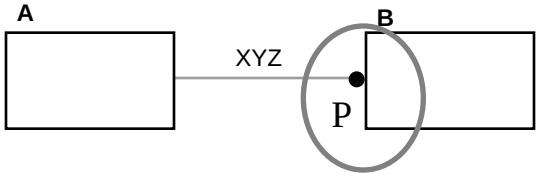
   If no, the A:B relationship is *"into"*...



2. Can there be an *A* which does not participate in an *XYZ* relationship with some *B*?
   If yes, the A:B relationship is *"partial"* (at most 1)...



   If no, the A:B relationship is *"total"* (exactly 1)...



Optional participation is an "analysts's clue." It suggests further investigation using techniques which examine behavior and subtyping. By separating the notions of cardinality and participation, the analyst is provided two separate concepts which can be dealt with at different points in time or levels of modeling. For example, participation may be ignored in

the Business-perspective model. As with cardinality, participation is generally expressed graphically at each end of the connector line.

In the following pages, each of the five basic relationship types will be described in terms of:

- CONCEPTS
- CARDINALITY
- PARTICIPATION
- OTHER CHARACTERISTICS
- RELATIONSHIP NAME
- REPRESENTATION (in terms of the alternative forms and perspectives)

4.1 ASSOCIATION RELATIONSHIPS

4.1.1 CONCEPTS

An association relationship is a business-based interaction between entities.[3] The relationship is named, typically as a verb phrase, and defined explicitly in terms of both cardinality and participation.

Some modeling techniques allow the expression of *n*-ary relationships (relationships with more than 2 entities participating). Most simply, an *n*-ary relationship may be thought of as a bundle of individual binary pairings. Thus, a 3-ary relationship (e.g., the ASSIGNMENT relationship between PROJECT, ROLE, and PERSON) can be represented as the set of its binary relationship pairs, each with its own cardinality and participation characteristics (see example in 4.1.2.1). More recent approaches have challenged this technique as too simplistic ~~ that the complexity of the n-ary relationship is more than the mere sum of its binary parts. For example, the Snetskaya cardinality approach selects each entity in turn and expresses its cardinality to the remaining set of entities in the relationship.

4.1.1.1 CARDINALITY

The cardinality of an association relationship can be any combination, i.e., 1:1, 1:M, M:1, or M:M.

4.1.1.2 PARTICIPATION

An association relationship may be declared optional in any direction.

4.1.1.3 RELATIONSHIP NAME

Association relationship names are expressed as a "verb form," either as a transitive verb (with the two connected entities serving as the "subject" and "object"):

CUSTOMER	***RESERVES***	TABLE
<entity>	<relationship>	<entity>

or as a noun form, with no directionality implied:

CUSTOMER	***RESERVATION***	TABLE
<entity>	<relationship>	<entity>

In the former case, the relationship name implies some "directionality." Note that it could also be expressed as "TABLE is reserved for CUSTOMER." When the verb-phrase form is used, the notion of a "subject" (anchor) entity is stored and related to a verb phrase relationship name. Some approaches also store a second verb phrase for the relationship name when read "in reverse."

---

[3] generally between different entity types, although "self-to-self" associations are allowed with a single entity type.

The noun form does not require that directionality be explicitly stated or stored. However, translating between the graphic and the structured English forms is more difficult using the noun-form approach. It is typically vocalized as "There is a RESERVATION relationship between a CUSTOMER and a TABLE" – which makes the data analysis concepts more visible in the somewhat artificial grammar.

4.1.2 REPRESENTATION OF ASSOCIATION RELATIONSHIPS

4.1.2.1 STRUCTURED ENGLISH

The following structured English syntax expresses the cardinality, participation, and name properties of an association type relationship.

Syntax:

<**relationship_name**>
Each <entity_name_a> {always|sometimes} <verb_a> <cardinality_b> <entity_name_b>[s].
Each <entity_name_b> {always|sometimes} <verb_b> <cardinality_a> <entity_name_a>[s].

Examples:

RESERVATION relationship:
Each CUSTOMER always reserves 1 or more TABLEs.
Each TABLE sometimes is reserved for 1 or more CUSTOMERs.

In the simple (set of binary pairs) form:

ASSIGNMENT relationship:
Each PROJECT always requires 1 or more ROLEs.
Each ROLE always is required by 1 or more PROJECTs.

Each ROLE sometimes is filled by 1 or more PERSON.
Each PERSON always is qualified as 1 or more ROLEs.

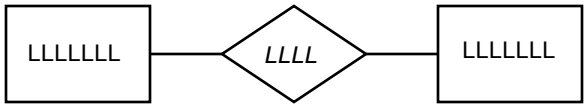
Each PERSON sometimes is assigned to 1 or more PROJECTs.
Each PROJECT sometimes staffs 1 or more PERSONs.

The GUIDE BSPI Methodology project adopted the term *Semantic Data Unit (SDU)* to describe these component entity pairings within an association relationship definition.

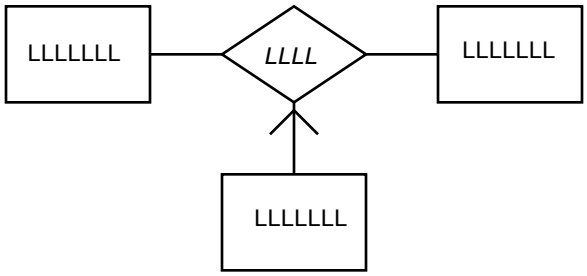
4.1.2.2 GRAPHIC ALTERNATIVES

4.1.2.2.1 BUSINESS PERSPECTIVE MODEL (ER/IM)

Business-perspective modeling allows both binary and n-ary association relationships. However, an association with more than 4 or 5 entities is a clue to the data analyst that the relationship could bear further scrutiny.

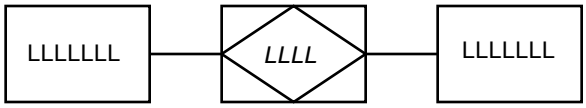


- relationship is indicated by a DIAMOND shape
- assumed M:M unless explicitly annotated
- optionality is not expressed graphically
- connectors are not labeled



- relationships can have their own attributes; furthermore a relationship may "grow" in significance to the point where it becomes an entity. This is indicated by the GROWS symbol on the line.
- this also occurs when a given relationship needs to participate in a relationship

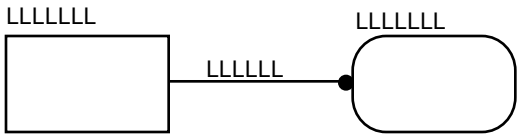
An alternative form draws the associative entity box around the relationship:



4.1.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL (DMT)

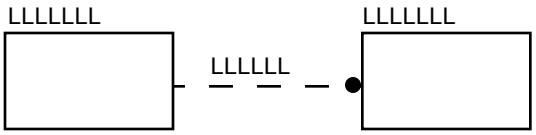
At the I/S-perspective, all association relationships are binary. Various symbol and connector graphics are used to express additional semantics.

DMT Identifying Relationship



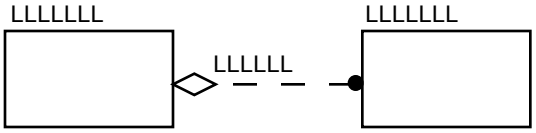
- PUFFY BOX indicates an entity which cannot exist in the absence of the relationship
- relationship name on the line is optional, except when there are 2 or more relationships between the same 2 entities
- BIG DOT indicates a "many" cardinality

DMT Non-Identifying Relationship



- both entities can exist independently in the absense of a relationship
- BIG DOTs at either end would represent "many" cardinality

DMT Optional Relationship

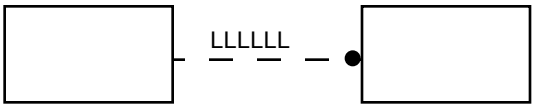


- the SMALL DIAMOND means the identifier of the entity touched by the diamond may be null in the related entity (i.e., the foreign key)
- by definition, only a non-identifying relationship can be optional

The participation of the association in the "other direction" (toward the big dot) is assumed to be optional. If it is mandatory, a "P" is annotated adjacent to the big dot.

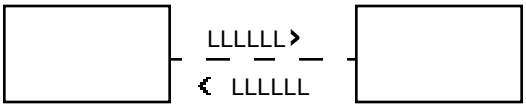
DMT Extensions for Relationship Name Representation

At the I/S-perspective, occasionally the relationship name is shown on the graphic model. This name label may also indicate its "directionality."



- the relationship name is "read" toward the big dot.

Non-DMT approaches may graphically depict the relationship names in both directions. In that case, some additional visual clue is used to indicate the "directionality" of the relationship phrase.



- relationship labels indicating "directionality" in both directions

4.1.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

The relational model does not have an explicit way of designating an association relationship between tables. Beginning in DB2 Ver. 2, these semantics will be more explicitly declared by allowing Primary Key/Foreign Key relationships to be defined in the table definitions.

REPRESENTATIVE

REP	NAME
R01	Michaelson
R20	Jackson
R45	Handler
R23	Anderson
R88	Smith

- connectors are Foreign Keys (a join over a common domain)
- connectors are not graphically depicted

CUSTOMER

ID	NAME	CITY	REP
C1	James	Seattle	R23
C2	Kent	Bellevue	R88
C4	Cushing	Olympia	R23
C9	Nelson	Kent	R01
C11	Hays	West Seattle	R20
C21	Miles	Seattle	R20
C70	Jones	Winslow	R01

4.1.3 META-MODEL TO REPRESENT ASSOCIATION RELATIONSHIPS

RELATIONSHIP::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::=<tag[?]>; <id_reference>;<text>]
* **SD**U::=<tag>;<id_entitya>;<id_entityb>; [{M|1|n}] [{R|O|C|F }] [: {M|1|n}] [{R|O|C|F }];
<verba>[;<verbb>]

* Multiples allowed

4.2 GENERALIZATION RELATIONSHIPS

4.2.1 CONCEPTS

One of the fundamental extensions to early modeling techniques which has now been adopted by virtually every modeling approach is some form of "generalization" relationship. A generalization relationship relates a *specific* type to a more *generic* type. The inverse of generalization is referred to as "specialization" (from generic to specific). [TSIC82] Generalization allows objects to be (conceptually) broken up into more specialized objects. For example, the notion of an EMPLOYEE can be broken into more specific entities as SALARIED EMPLOYEE and HOURLY EMPLOYEE. Inversely, an HOURLY EMPLOYEE and a SALARIED EMPLOYEE can both be generalized into the common concept of EMPLOYEE. (I say that these are "conceptually" broken up because, remember, an instance is still one instance!)

This relationship type is called by a variety of different names, for example:

- MULTITYPING
- SUPERTYPE / SUBTYPES
- INHERITANCE HIERARCHY
- CATEGORIZATION STRUCTURE
- CLASSIFICATION STRUCTURE

Regardless of label, all share the notion of "inheritance" (the subordinate entity is an instance of its "parent" and therefore inherits its attributes). Some techniques go on to allow subtypes to express explicit "overrides" to inherited traits. When the notion of operations or actions is added to the generalization structure, the data model starts to take on "object oriented" qualities.

Membership in a generalization group is considered to be exclusive (disjoint), i.e., an instance can never be more than one of the subtype entity types. If it can be, there must be a second, separate generalization group for the supertype entity. I have found no papers that attempt to describe constraint rules for membership **between** multiple generalization groups of a single supertype entity.

It is more precise to give an explicit name to the generalization group (set) of which the subtypes are members. When this is done, and when attributes of the CLASS itself are kept, the CLASS entity becomes an entity on its own right, and the CLASSIFICATION RELATIONSHIP is made explicit. This often helps to emphasize that the CLASS entity is not the same as one of the SUBTYPE entities. (*"The set of walruses is not a walrus."* – GEB). For example, when categorizing HUMAN into FEMALE and MALE, the generalization (CLASS) set is GENDER. However, this notion of set-ness is not always a well-formed business concept so in practice this part of the discipline is frequently omitted from the data model.

4.2.1.1 CARDINALITY

The cardinality of a generalization relationship is always 1:1 (one instance of a subtype is always the same instance as its supertype).

4.2.1.2 PARTICIPATION

A generalization relationship is actually a relationship between a "super" entity type and a set of "sub" entity types. Therefore, the participation designation is more relevant when viewed in terms of the supertype's relationship to the set as a whole. For example, it can be ***exhaustive*** (an instance of the supertype must be represented by an instance of one and only one of the listed subtypes) or ***limited*** (an instance of the supertype can be represented by an instance of one or none of the listed subtypes – in other words, the subtypes identified in the model do not represent an enumerated list of subtypes).

An instance of a subtype entity always implies a mandatory relationship to its supertype.

4.2.1.3 OTHER CHARACTERISTICS OF GENERALIZATION RELATIONSHIPS

In addition to cardinality and optionality, generalization relationships may also be distinguished as either STATIC or TIME-VARYING. An example of STATIC subtyping is:

MALE	or	FEMALE	***IS-A***	PERSON
<subtype>		<subtype>	<relationship>	<supertype>

A *static* subtype tends to stay in its designation over time.

An example of *time-varying* subtyping is:

PROSPECT	or	CURRENT_CUST	or	FORMER_CUST	***IS-A***	CUSTOMER
<subtype>		<subtype>		<subtype>	<relationship>	<supertype>

In this case, a given instance may move between the various subtype "states" over time. Techniques exist for specifying the valid state transitions in terms of the entity model, generally with notational extensions to the modeling vocabulary (e.g., ref. [LEAR87]). Such techniques are beyond the scope of this paper.

4.2.1.4 RELATIONSHIP NAME

This is generally called the ***IS-A*** relationship – every FEMALE *"**IS A**"* PERSON. In the reverse direction (supertype to subtype), the relationship name might be expressed as *"**CAN BE**."*

4.2.2 REPRESENTATION OF GENERALIZATION RELATIONSHIPS

4.2.2.1 STRUCTURED ENGLISH

The following structured English syntax expresses the cardinality, optionality, and name properties of a generalization type relationship.

Syntax:

Each <name_supertype_entity> ...
… {may|must} be categorized as a type of <name_category_class>, i.e.,
{may|must} be [one of]
a[n] <name_category_entity_a>,
[or a[n] <name_category_entity_b>,...]
depending on <name_attribute_category_discriminator>.

Each <name_category_entity> ...
… is a type of <name_category_class> and is a <name_supertype_entity>,
distinguished by <name_attribute_category_discriminator> of "<value>."

Examples:

Each TABLE ...
… must be categorized as a type of RESERVED FACILITY, i.e.,
must be one of
a RESERVED TABLE,
or an AVAILABLE TABLE,
depending on RESERVATION STATUS.

Each RESERVED TABLE ...
… is a type of RESERVED FACILITY and is a TABLE,
distinguished by RESERVATION STATUS of "reserved."

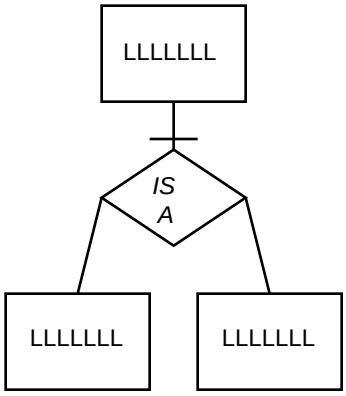
Each AVAILABLE TABLE ...
… is a type of RESERVED FACILITY and is a TABLE,
distinguished by RESERVATION STATUS of "available."

Note: a supertype entity may itself be a category entity if it has a "parent" supertype.

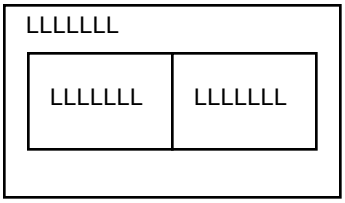
4.2.2.2 GRAPHIC ALTERNATIVES

4.2.2.2.1 BUSINESS PERSPECTIVE MODEL (ER/IM)

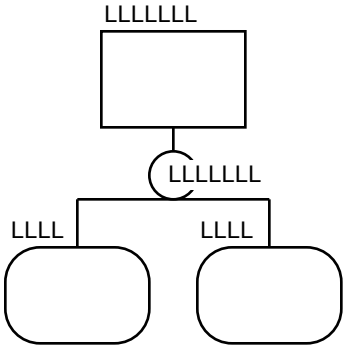
Information Modeling Representation



Variation Used in GUIDE BSPI Data Model

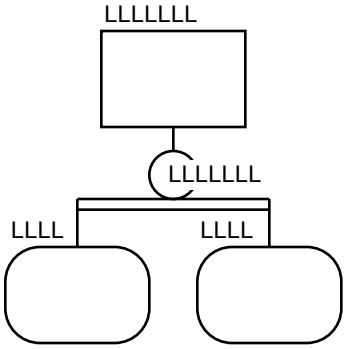


4.2.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL (DMT)



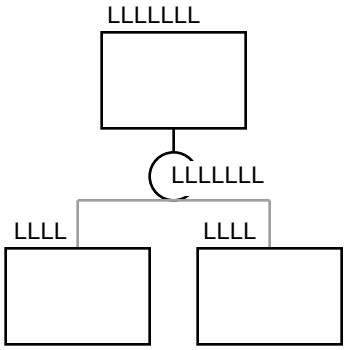
- supertype entity
- category discriminator attribute name
- category set
- subtype entities (members of the set)

Exhaustive participation can be graphically depicted by drawing a double line under the category discriminator symbol.



- an exhaustive category set

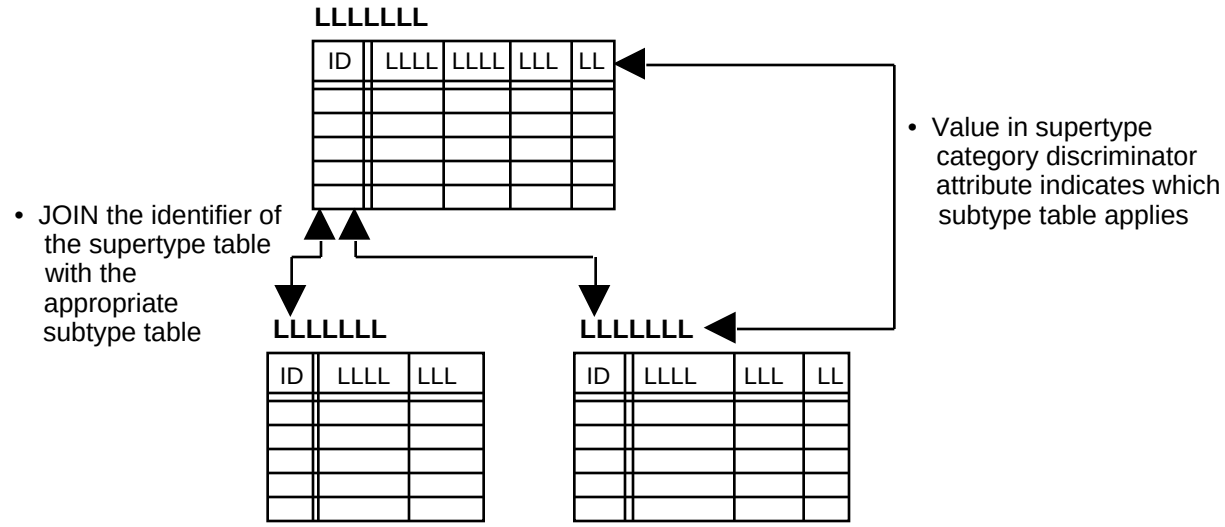
Since the subtypes are most frequently assumed to inherit the identifier of the supertype (along with the other supertype attributes), the puffy box shape and solid line graphics are the most commonly seen. However, in the case of a subtype which does not share its identifier with its supertype generic parent, the same visual notations for expressing these semantics are used. For example,



- subtypes which do not use the same identifier

4.2.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

The relational model does not have an explicit way of designating a generalization relationship between tables. In DB2, these semantics could be implemented through Catalog extensions or by using VIEW mechanisms to perform the JOIN, creating a single VIEW table that inherits the attributes of the parent base table/view and appends the unique subtype attributes or Foreign Key relationships.



This solution suffers from the need to perform the JOIN and, with today's products, the fact that the application code must bear the burden of relating the value of an attribute (the category discriminator) with the name of a data object (the subtype table).

An alternative solution would place all of the attributes from the subtypes into the generic parent table, allowing NULLS. However, the application code would need to enforce the either/or nature of these attributes making the semantics of the model much less explicit.

4.2.3 META-MODEL TO REPRESENT GENERALIZATION RELATIONSHIPS

RELATIONSHIP::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::=<tag[?]>; <id_reference>;<text>]
GENERALIZATION::=<id_generalization_entity>;<id_attribute>[;<id_class_entity>]
;<valueset> [; { L |E}] [;{S|D}]
* **CA**TEGORY::=<id_category_entity>;<value>

* Multiples allowed

4.3 ROLE-PLAYING RELATIONSHIPS

4.3.1 CONCEPTS

A role-playing relationship relates a collection of different entity types. At first glance, this type of relationship appears quite similar to a generalization relationship. It is distinguished most significantly by a de-emphasized notion of *inheritance* in favor of the concept of *role playing.* For example, in our company a PROJECT MANAGER can be either an EMPLOYEE or a CONTRACTOR. However, since not all EMPLOYEEs (or all CONTRACTORs) are PROJECT MANAGERs, EMPLOYEE cannot be considered a subtype of PROJECT MANAGER. Also, the PROJECT MANAGER entity would not be thought of as inheriting all the attributes of both the EMPLOYEE and CONTRACTOR entities. Because PROJECT MANAGER is a *generalized role* that either an EMPLOYEE or a CONTRACTOR may play, it is treated as a role-playing relationship rather than a generalization relationship. [SCHU87] Another indicator of a role-playing relationship (vs. a generalization relationship) is the identification of the participating entity types with different identifier schemes.

4.3.1.1 CARDINALITY

The cardinality of a role-playing relationship is always 1:1 (the role-playing entity is always an instance of one of its "source" entity types).

4.3.1.2 PARTICIPATION

An instance of the role-playing entity must have one and only one source entity instance. An instance of the source entity does not necessarily imply an instance of the role-playing entity.

4.3.1.3 RELATIONSHIP NAME

This can be called the ***PLAYS-ROLE-AS*** relationship – an EMPLOYEE can ***PLAY ROLE AS*** a PROJECT MANAGER. In the reverse direction (role to source), the relationship name might be "***IS A ROLE PLAYED BY***" as in the example above.

4.3.2 REPRESENTATION OF ROLE-PLAYING RELATIONSHIPS

4.3.2.1 STRUCTURED ENGLISH

The following structured English syntax expresses the cardinality, optionality, and name properties of a role-playing type relationship.

Syntax:

Each <name_role-playing_entity> ...
… is a role that can be played by a[n] <name_source_entity>{.|,}
[or by a [n] < name_source_entity>{.|,}...]

Each < name_source_entity > ...
… can play the role of < name_role-playing_entity >.

Examples:

Each PROJECT MANAGER ...
… is a role that can be played by an EMPLOYEE,
or by a CONTRACTOR.

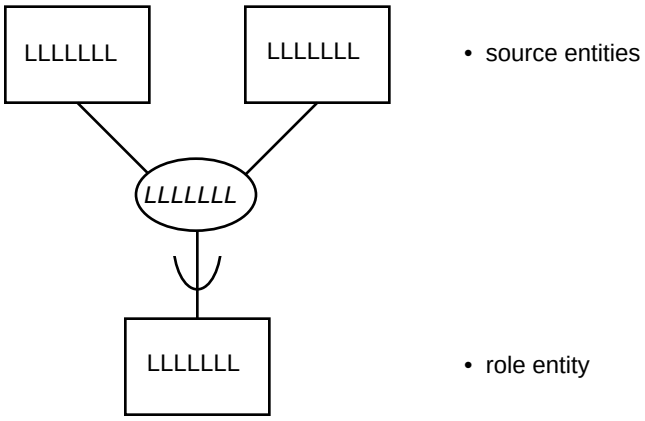
Each EMPLOYEE ...
… can play the role of PROJECT MANAGER.

Each CONTRACTOR ...
… can play the role of PROJECT MANAGER.

4.3.2.2 GRAPHIC ALTERNATIVES

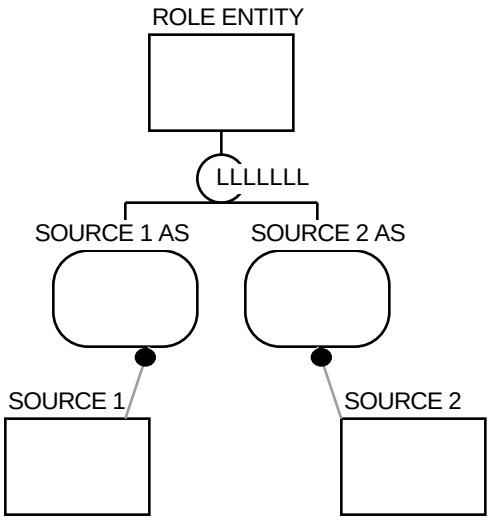
4.3.2.2.1 BUSINESS PERSPECTIVE MODEL (ER/IM)

The *BUSINESS* perspective model is typically not called on to reflect the detail of *roles* (or, to depict that distinction graphically). In those cases where it is, the following Information Modeling extension has been adopted. [SCHU87]



4.3.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL (DMT)

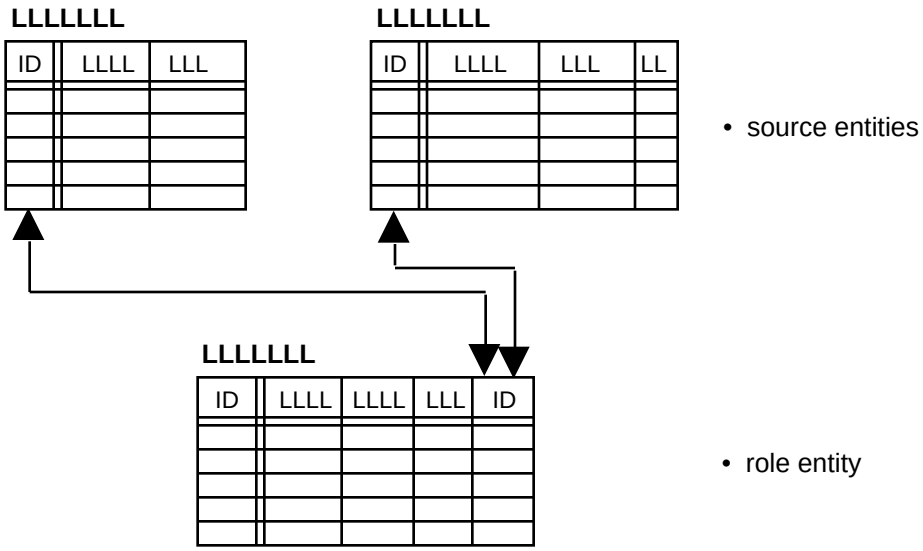
Currently DMT notation does not recognize a separate semantic construct to represent a role-playing relationship. Instead, the existing constructs of generalization, with relationships at the category entity perspective, would be used to implement this type of relationship.



Others would argue for the ability to declare this type of relationship, perhaps as a special type of generalization. It would simplify the graphic representation of the model and allow the particular semantics of this type of relationship to be expressed more tersely.

4.3.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

The relational model does not have an explicit way of designating a role-playing relationship between tables, and it represents a theoretical problem for the relational model in general since it would require some notion of an "either-join" (join a table to another table depending on the value of the identifier or some other control attribute ~~ something not in happy agreement with the mathematical basis of the relational model!) Thus, for those who do implement this type of relationship today, the burden falls to the application code.



4.3.3 META-MODEL TO REPRESENT ROLE-PLAYING RELATIONSHIPS

RELATIONSHIP::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::=<tag[?]>; <id_reference>;<text>]
ROLE**P**LAYING::=<id_role_entity>
* **RO**LESOURCE::=<id_source_entity>

* Multiples allowed

4.4 AGGREGATION RELATIONSHIPS

4.4.1 CONCEPTS

Some entities may be related in an aggregation relationship, either as the aggregate entity (something made up of other entities) or as a component (constituent) entity. And, as in the classic "bill of materials" structure, there is nothing to prohibit a component from itself being an aggregate.

How is an aggregation relationship different from a simple association relationship where the relationship name is something like *CONTAINS* or *IS MADE OF*? Would "ENGINE *CONTAINS* OIL" be an aggregation or an association? How about "CHAIR *IS MADE OF* a BACK, a SEAT, and four LEGs"? The distinction may be largely subjective (which is perhaps why many modeling methodologies choose to ignore aggregations as a distinct concept). The Smith & Smith articles [SMIT77],[SMIT77B] are the classic sources on the topic for the 'true believer.'

For a simple heuristic, I like to think in terms of the instances and ask *"how many things do I think of when I think of this object?"* If the answer is *"it depends; sometimes it's one thing and other times (or to other people) it's several"* then that's a good clue that you are on the road to an aggregate. (Personally, I never think of the ENGINE and the OIL as a single object, even on that rare occasion when I participate in the *CONTAINS* relationship!) One other heuristic is to ask if the aggregate can exist without its components. (The engine and the oil can exist separately, but the chair loses its integrity as a chair if it is short a seat.) A last heuristic is to ask if the aggregate entity has properties of its own that are distinct from the summary attributes of its components (an example is given below).

Aggregates can be simple (homogeneous ~~ made up of only a single entity type) or complex (heterogeneous ~~ made up of several different entity types). I'm still looking for good examples – the best I have on hand at the moment are:

- **SIMPLE**: BUSINESS UNIT GROUP is made of ORGANIZATIONAL UNITs. (from IBM's BSPI Methodology)

- **COMPLEX**: CLASSROOM is made up of desks, chairs, an instructor's desk, a black/white board, and sometimes AV equipment.

I hesitate to talk about ORGANIZATIONAL UNITs in this example. It can introduce a degree of confusion because an organizational unit can be related to another organizational unit either in a self-to-self association relationship or in an aggregation relationship (or sometimes both equivocally). Again, I resort to the *"how many are there?"* test. If organizational units AA, AB, and AC "report to" organizational unit XY, how many employees are in organizational unit XY? If the answer is "the sum of the employees in AA, AB, and AC, plus any administrative/staff employees at the XY perspective," then ORGANIZATIONAL UNIT is being treated as an aggregate. If, on the other hand, the employees in XY are only those directly assigned to XY, then it's an association relationship.

(Trying this with process decomposition structures vs. a HIPO diagram is left as an exercise for the reader.)

Finally, it is advisable to be aware that the concept of "aggregation" is used variously by different modelers. Some use this term to refer simply to any generalized abstraction technique which is used (for a moment) to hide certain details on a model and emphasize others. I use "VIEWS" for this type of abstraction [ref. Section 6] and in this topic will only discuss the more narrow definition of aggregation as a specialized relationship type.

4.4.1.1 CARDINALITY

The cardinality of an aggregation relationship is usually 1:M (aggregate to component). However, it is possible for 1:1 (where the aggregate consists of more than one entity type) and M:M (where the aggregate is "conceptual" rather than "physical").

4.4.1.2 PARTICIPATION

Any aggregation relationship may be declared optional in either direction.

4.4.1.3 RELATIONSHIP NAME

This is generally called the ***CONSISTS-OF*** relationship – every chair consists of seat, legs, etc. In the reverse direction (component to aggregate), the relationship name might be *"**IS A PART OF"*** as in the example below.

4.4.2 REPRESENTATION OF AGGREGATION RELATIONSHIPS

4.4.2.1 STRUCTURED ENGLISH

The following structured English syntax expresses the cardinality, optionality, and name properties of an aggregation type relationship.

Syntax:

Each <name_aggregate_entity> ...
… is an aggregate {always|sometimes} made up of <cardinality> <name_comp_entity>[s]{.|,}
[and {always|sometimes} of <cardinality> <name_comp_entity>[s] {.|,}
...]

Each < name_comp_entity> ...
… {is|can be} a part of <cardinality> < name_aggregate_entity >.

Examples:

Each CHAIR ...
… is an aggregate always made up of 1 SEAT,
and always of 3-4 LEGs,
and always of 1 BACK,
and sometimes of 2 ARMs.

Each SEAT ...
… is a part of only 1 CHAIR.

Each LEG ...
… is a part of only 1 CHAIR.

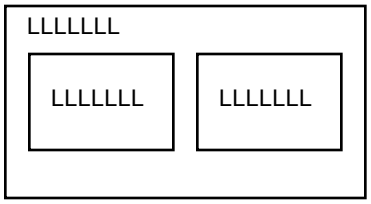
Each BACK ...
… is a part of only 1 CHAIR.

Each ARM ...
… is a part of only 1 CHAIR.

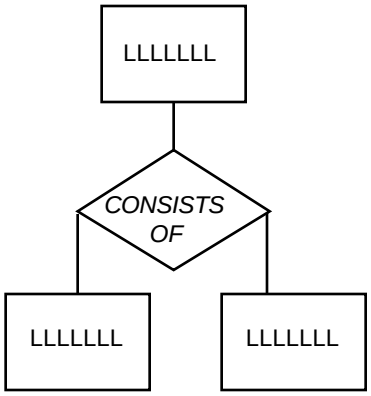
4.4.2.2 GRAPHIC ALTERNATIVES

4.4.2.2.1 BUSINESS PERSPECTIVE MODEL (ER/IM)

The *BUSINESS* perspective model does not generally reflect the detail of the constituents of an aggregate. If it is called upon to reflect this perspective of detail, I have used the following representation, adapted from the GUIDE Repository Data Model.



Another representation technique applied at this perspective is simply to use the basic relationship notation with a *CONSISTS OF* relationship name. Note that this does not graphically distinguish the aggregate entity from its components.

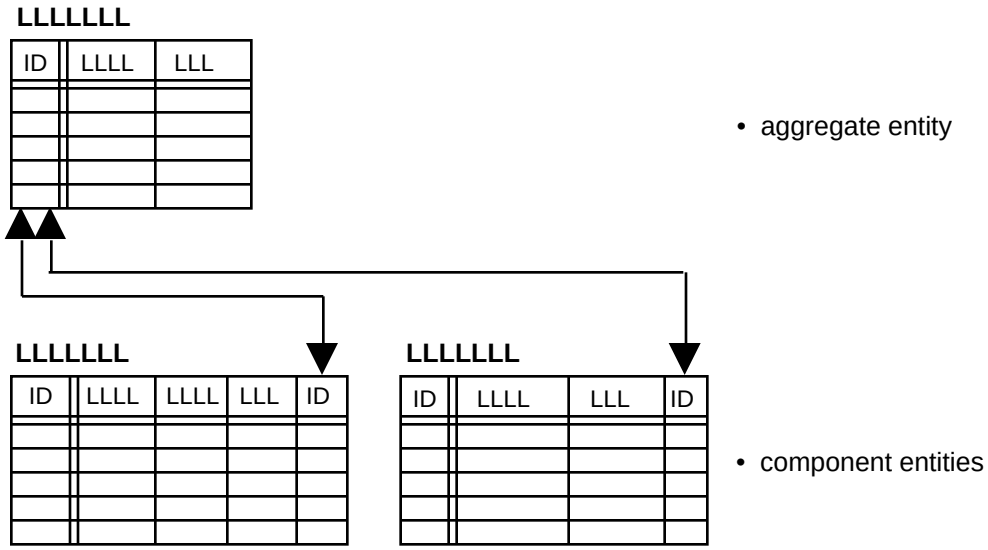


4.4.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL (DMT)

Again, the documented DMT notation does not have a separate construct to represent an aggregation. The usual association relationship constructs are used; the aggregate is one entity type and the constituents are separate entity types (or in the case of a simple, hierarchical aggregation, a single entity type with a self-referring Foreign Key). However, this again loses some of the expressiveness that a separate, explicitly declared relationship type would give. For example, in the case of an organizational unit entity with an association to itself, we would not be able to distinguish whether the business rule behind the EMPLOYEE_COUNT of an organizational unit was merely the count of the employees in the single unit or the total of all those “rolling into” the entity instance.

4.4.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

The relational model does not have an explicit way of designating an aggregation relationship between tables. In DB2, these semantics could be implemented through Catalog extensions or application code. In other products (e.g., Oracle), language extensions have been made to SQL to allow the DML to navigate these possibly recursive structures. However, even this does not move the expression of the semantics into the declared (DDL) portion of the data definition.



4.4.3 META-MODEL TO REPRESENT AGGREGATION RELATIONSHIPS

RELATIONSHIP::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::=<tag[?]>; <id_reference>;<text>]
AGGREGATE::= <id_aggregate_entity> [; {S [B|D]|C}]
* **PA**RTOF::=<id_component_entity> ; [{1|M|n}] [{R|O|C}] [: {1|M|n}] [{R|O|C}]

* Multiples allowed

4.5 CHARACTERISTIC RELATIONSHIPS

4.5.1 CONCEPTS

The characteristic relationship treats a set of characteristics (properties or attributes) of an entity as an entity type in its own right. Since ENTITY *HAS* ATTRIBUTE is a special type of relationship (discussed in Section 5), why then is it discussed as a type of relationship between entity types? When is something an entity and when is it an attribute? (To quote a Data Modeling Truism: *"One person's entity is another person's attribute."*)

The general case calling for a characteristic relationship:

- when an entity has multi-valued properties.

In FLAV81, a second case ~~ when a property of an entity itself takes on properties ~~ was also identified as a "*characteristic*." As the modeling discipline has matured since that original work, this case has tended to be treated under the topic of "*classification*." Thus, this second variation of *characteristic* will not be discussed in this section.

Value-dependency is the distinguishing test for a characteristic relationship. [FLAV81] For single-valued attributes, a value-dependency is referred to as a "functional dependency" and such attributes are merely associated with the basic entity. (All the attributes of an entity are value dependent on the entity's identifier – and therefore are in at least third normal form.) But, what does the analyst do with any multi-valued attributes of an entity – when an entity's identifier determines a set of values of a single attribute? When for each distinct instance of an entity there is a finite, unique set of values for a property, each instance of that set is treated as a characteristic entity joined to its owner via a characteristic relationship type. Modeled properly this ensures that the data is in at least 4th normal form.

An example is the entity EMPLOYEE, with attributes NAME, PHONE, TITLE, and SECURITY CLEARANCE. If the business rules indicate that a employee may have several SECURITY CLEARANCE. Also, each of those can itself have attributes such as CLEARANCE TYPE CODE, AUTHORITY LEVEL, DATE GRANTED, and RESPONSIBLE OFFICER. (Note: this is distinct from the SECURITY CLEARANCE TYPE entity which might have attributes such as CLEARANCE TYPE CODE and CLEARANCE TYPE DESCRIPTION.)

One special comment – in developing a data model at the Business or the I/S perspective, classic data modeling techniques suggest that the analyst ignore the time dimension in expressing the data requirements. However, it is not uncommon to find a model depicting time-varying entities such as historical information (for example, in capturing various results of lab tests performed at different points in time.) In this case, time-dependent descriptive information is treated as a characteristic entity type. Considerable attention is currently being given to the "temporal dimension" of a data model as a separate and distinct notion, but until these concepts are settled, the characteristic relationship will carry the weight of modeling time semantics.

4.5.1.1 CARDINALITY

By definition, the cardinality of a characteristic relationship is 1:M. There are some approaches cases which challenge the analyst to seek out "the other" identifying parent (generally some form of "type code"), resulting in *characteristic* relationships which look like *associations.* However, while this can be looked on as a good heuristic for testing the completeness of a logical design, it loses some of the business-focused expressiveness gained by allowing an entity to be viewed as the whole of both its single- and multi-valued properties.

4.5.1.2 PARTICIPATION

A characteristic relationship may be declared optional only in the direction of the owner to the characteristic entity. For example, the EMPLOYEE may not have any DEPENDENTs. However, in the context being modeled, a DEPENDENT may not exist without a relationship to an EMPLOYEE.

4.5.1.3 RELATIONSHIP NAME

This could be called the ***HAS*** relationship – an EMPLOYEE ***HAS*** DEPENDENTs. In the reverse direction (characteristic to owner), the relationship name might be *"**BELONGS TO"*** as in the example above.

4.5.2 REPRESENTATION OF CHARACTERISTIC RELATIONSHIPS

4.5.2.1 STRUCTURED ENGLISH

The following structured English syntax expresses the cardinality, optionality, and name properties of a characteristic type relationship.

Syntax:

Each <name_owning_entity> ...
… {sometimes|always} has <cardinality> <name_category_entity>[s].

Each < name_category_entity> ...
… always belongs to only 1 < name_owning_entity>[s].

Examples:

Each EMPLOYEE ...
… sometimes has 1 or more DEPENDENTs.

Each DEPENDENT ...
… always belongs to only 1 EMPLOYEE.

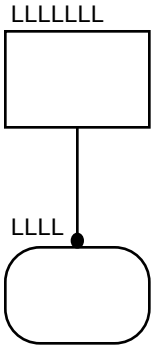
4.5.2.2 GRAPHIC ALTERNATIVES

4.5.2.2.1 BUSINESS PERSPECTIVE MODEL (ER/IM)

The *BUSINESS* perspective model does not express this perspective of detail in an explicit graphic construct. In fact, Information Modeling defines the special concept of "Entity Normal Form," which (informally) is the notion of an entity being in Third-Normal Form but not in First-Normal Form. By this definition, all attributes of an entity are dependent on the entity's identifier (and not on any other better identifier). In attribution the analyst places each attribute in its "single best home," regardless of whether the attribute is single- or multi-valued in that home.

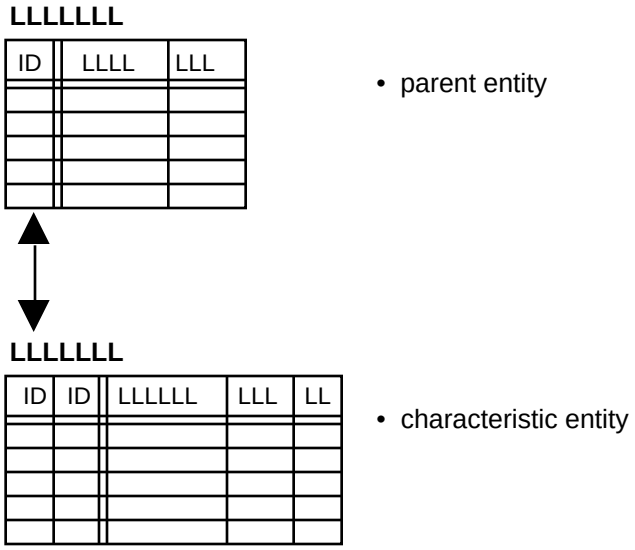
4.5.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL

In DMT, the characteristic entity is simply depicted as an identity-dependent, existence-dependent entity which has only one identifying parent.



4.5.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

The relational model does not have an explicit way of designating a characteristic relationship between tables. In DB2, these semantics could be implemented through Catalog extensions.



4.5.3 META-MODEL TO REPRESENT CHARACTERISTIC RELATIONSHIPS

RELATIONSHIP::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::=<tag[?]>; <id_reference>;<text>]
CHARACTERISTIC::=<id_parent_entity>;<id_characteristic_entity> [;1: {M|1|n}] [{R|O|C}]

* Multiples allowed

5. ATTRIBUTES

5.1 CONCEPTS

I cannot think about *attributes* without thinking of Bill Kent and Dan Tasker, who think about attributes a lot more than I do. It is perhaps easier to illustrate the concept of attribute rather than give it a rigorous definition:

> *Lots of things have lots of attributes. People have heights and birthdays and children, my car is blue, and New York is crowded. Much of the information in an information system records the attributes of things.* [KENT78]

Attribution is one of those concepts which get fuzzier rather than clearer if you dwell on them too long. If you do want to test your conceptual grasp, I suggest a chilly winter evening, a warm fire, a blanket, and a copy of Data And Reality. Then, after you've pondered over the philosophical questions raised by Kent, you can look to 4th Generation Data for some concrete, practical answers [TASK89]. For the scope of this paper, I will introduce only the basics about attributes. Most simply, an attribute can be defined as *a named characteristic or property of an entity (or relationship)*. In Kent's example above, PERSON is an entity with attributes such as HEIGHT, BIRTHDATE, and NAME OF CHILD.

In an entity instance these attributes are considered to take on one or more values. For example, my HEIGHT (an attribute of a specific instance of the entity PERSON) is *5'5"* (a value). This is an example of a *single-valued attribute*. When an attribute of a given entity can take on several values at the same time, it is considered a multi-valued attribute. The COLOR of my CAR is *blue* and *white.*

A concept which relates to (and is often confused with) attribute is that of *domain*.[4] A domain can be thought of as the pool (or *set*) of valid values from which an attribute can draw a value at a given point in time. The introduction of the concept of domain modifies the definition of attribute slightly to become *the association of an entity and an element of a domain*; attribute has also been described as the *role* a domain plays for a given entity. Frequently some form of role naming scheme is used in assigning a unique and meaningful name to an attribute. A good introductory reference is [CURT83]; much has been written since to further refine the concept of domain, especially as it is (or should be) implemented in support of the relational model (e.g., [POLL87]).

A related concept is that of *identifier*, often confused with the stricter term *key*. Since the concept of entity requires that each entity occurrence be uniquely distinguishable from all other entity occurrences, it is obvious that each entity must have at least one identifier. An identifier is considered to be one of the entity's attributes.

---

[4] And, it doesn't help that the term domain is used equivocally within the field of systems analysis. It is also used to refer to the "domain of information" – the bounds or scope of the system under consideration. I do not mean that kind of domain when I use the term!

5.2 REPRESENTATION OF ATTRIBUTES

5.2.1 STRUCTURED ENGLISH

Each <name_entity> ...
- … is uniquely identifiable by <name_attribute>[,<name_attribute>,...] [,|.]
  and is also [uniquely] identifiable by <name_attribute> [,<name_attribute>,...] [,|.]

- … has properties of:
  • <name_attribute> [<name_role>,...]

5.2.2 GRAPHIC ALTERNATIVES

5.2.2.1 BUSINESS PERSPECTIVE MODEL (ER/IM)

At the perspective of the *BUSINESS* model, the notion of attribute is not emphasized. Attributes are consciously allowed to remain somewhat fuzzy, corresponding with our own imprecise and personal ways of visualizing the world we see. Information on attributes is captured mainly to clarify the meaning or essence of an entity or relationship; when this is done, attributes are referred to as *nominal attributes*. It is useful to annotate attributes identified at the business perspective of modeling as single- or multi-valued, if this can be readily determined.

Some data modeling approaches call for the designation of a *business entity identifier* at this perspective. This can be a very useful business-perspective activity if the culture of the organization allows it. However, while identifier specification is a valuable step which can have arguable benefits for the enterprise (independent of any automation reasons), in the wrong climate this endeavor can bog down, or even kill, an information modeling effort. When business identifiers cannot be established, it is sufficient to acknowledge that each business entity is **in some way** identifiable to the business. Alternatively, the varying identifier schemes can be merely logged, without attempting to force the business community to acknowledge **the one** identifier scheme for each entity type. Pragmatically, if that entity is not to receive automation support, the mediation of how that entity is identified by various parties can be an entertaining but non-essential exercise for the data analyst.

I do not depict attributes on the graphic Business-perspective model. Some do, for example, representing attributes as bullet labels outside the entity or relationship symbol.

- NAME
- CITY
- REP

5.2.2.2 I/S CONCEPTUAL PERSPECTIVE MODEL (DMT)

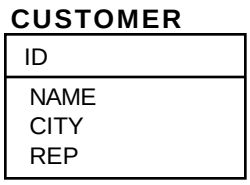
The I/S-perspective model can depict attributes in one of three modes:

- **no attributes displayed** – presenting a generalized model which focuses on the entities and their interrelationships

- **key / foreign key attributes only** – referred to as a key-based model. This model is typically used to represent a Data Architecture, or framework, for controlling the building of integrated data structures via individual, autonomous projects.

- **fully attributed model** – referred to as the Conceptual Data Model (CDM), logical data model, or logical database design. All of the attributes, and the primary key attributes, of the logical database design are depicted on the model. The model reflects at least 4th Normal Form.

The attribute names are listed inside the BOX shape, with the identifier attribute(s) shown above a horizontal line drawn across the box. Domains are not reflected explicitly on the model, other than possibly in the attribute naming scheme.



5.2.2.3 TECHNOLOGY PERSPECTIVE MODEL (RELATIONAL)

Attributes are shown as column headings on the instance table representation form of the model. Again, domains are generally not depicted graphically on the model.

CUSTOMER

ID	NAME	CITY	REP
C1	James	Seattle	R23
C2	Kent	Bellevue	R88
C4	Cushing	Olympia	R23
C9	Nelson	Kent	R01
C11	Hays	West Seattle	R20
C21	Miles	Seattle	R20
C70	Jones	Winslow	R01

5.2.3 META-MODEL TO REPRESENT ATTRIBUTES

```
        ATTRIBUTE::=<name>
            IDENTIFIER::=<id>
*           SHORTNAME::=<shortname>
*       [   SYNONYM::=<name>                                                    ]
*           DEFINITION::=<seq[?]>; <id_reference>;<text>
*       [   COMMENT::=<seq[?]>; <id_reference>;<text>                           ]
            DOMAIN::=<id_domain>
        [   ROLE::=<name>[;<name>... ]                                          ]
            PROPERTY::= {<id_entity>|<id_relationship>} [;P|A]
        [   PARTOF::=<id_attribute>                                             ]

        DOMAIN::=<name>
            IDENTIFIER::=<id>
            DOMAINTYPE::=<code>
*               INPUTAS::={Annn|Nnn.nn|D[J|MDY|DMY|YMD|C] [/|-]|T|LT}
*               STOREDAS::={Annn|Nnn.nn|D[J|MDY|DMY|YMD|C] [/|-]|T|LT}
*               VIEWEDAS::={Annn|Nnn.nn|D[J|MDY|DMY|YMD|C] [/|-]|T|LT}
*           [   VALUERULE::=<rule>                                              ]
*       [   VALUEPROC::=<procedure>                                             ]

*       Multiples allowed
```

6. VIEWS

Once a model grows past a digestible number of entities and relationships (about a dozen), it becomes too much for the mind to absorb all at once. The modeling device used to focus on a selected subset of an overall (*composite*) model while hiding the rest is referred to as a *view*.

Thus, a view is simply a defined scope of interest, either to a user or to an analyst. The view model reflects the underlying data requirements of that scope. The concept of view can be applied at all perspectives – Business, I/S, and Technology.

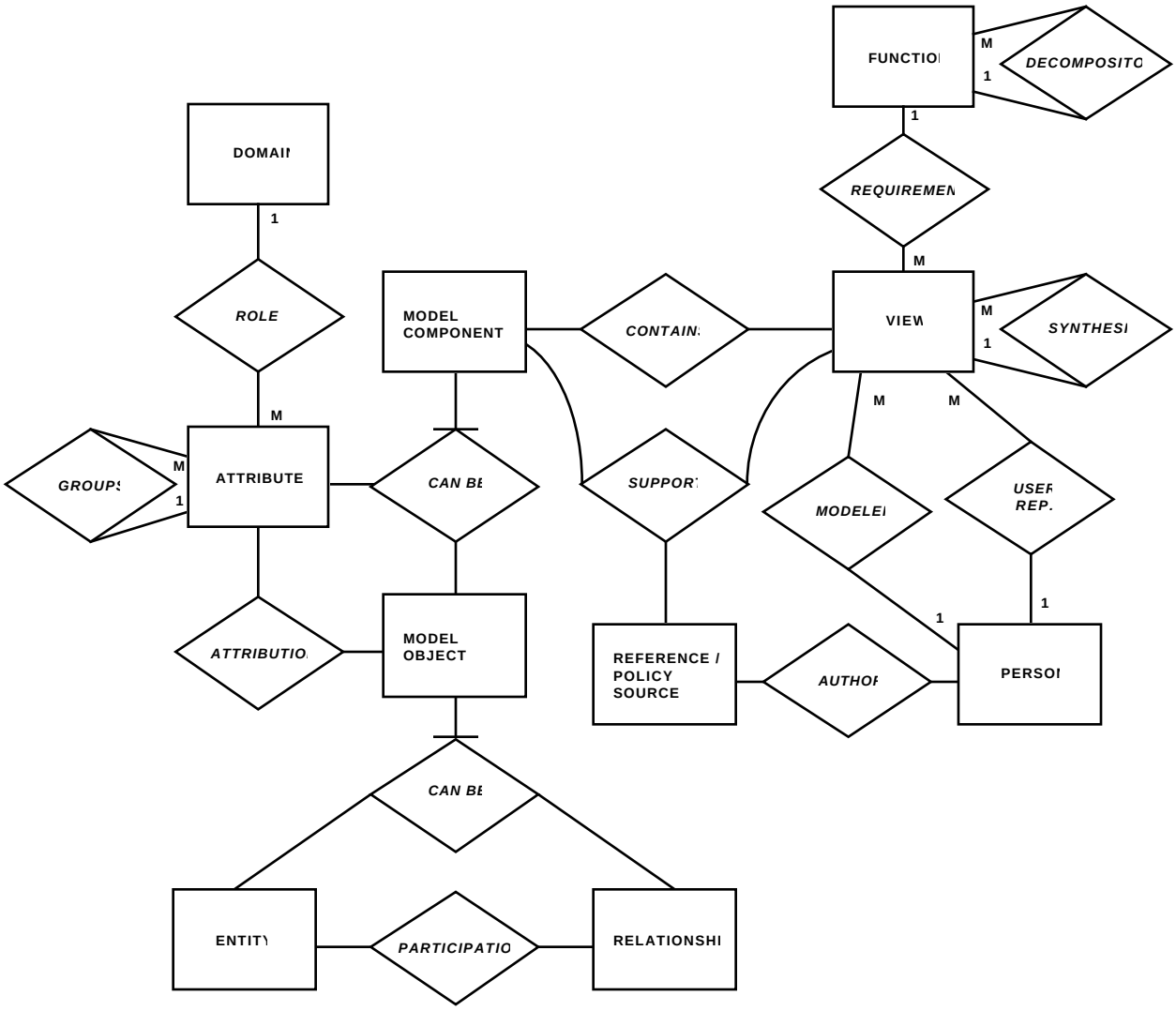
A data modeling tool should support the development of a composite data model from individual views. The entity / relationship / attribute definitions are shared between views but, especially in development, may reflect some slight inconsistencies from the perspectives of the individual views. The view model should be able to retain these inconsistencies in an unresolved state.

The data modeling tool should support **view reconciliation / view synthesis** as a discrete, analyst-selectable function. I am not of the camp that believes that view reconciliation should be forced by the tool as the model facts are entered. A view is a specific construct that endures through time; it does not disappear when the reconciliation is complete. (A view may, of course, be discarded at the discretion of the analyst.)

7. EXAMPLE SUPPORTING META-MODEL

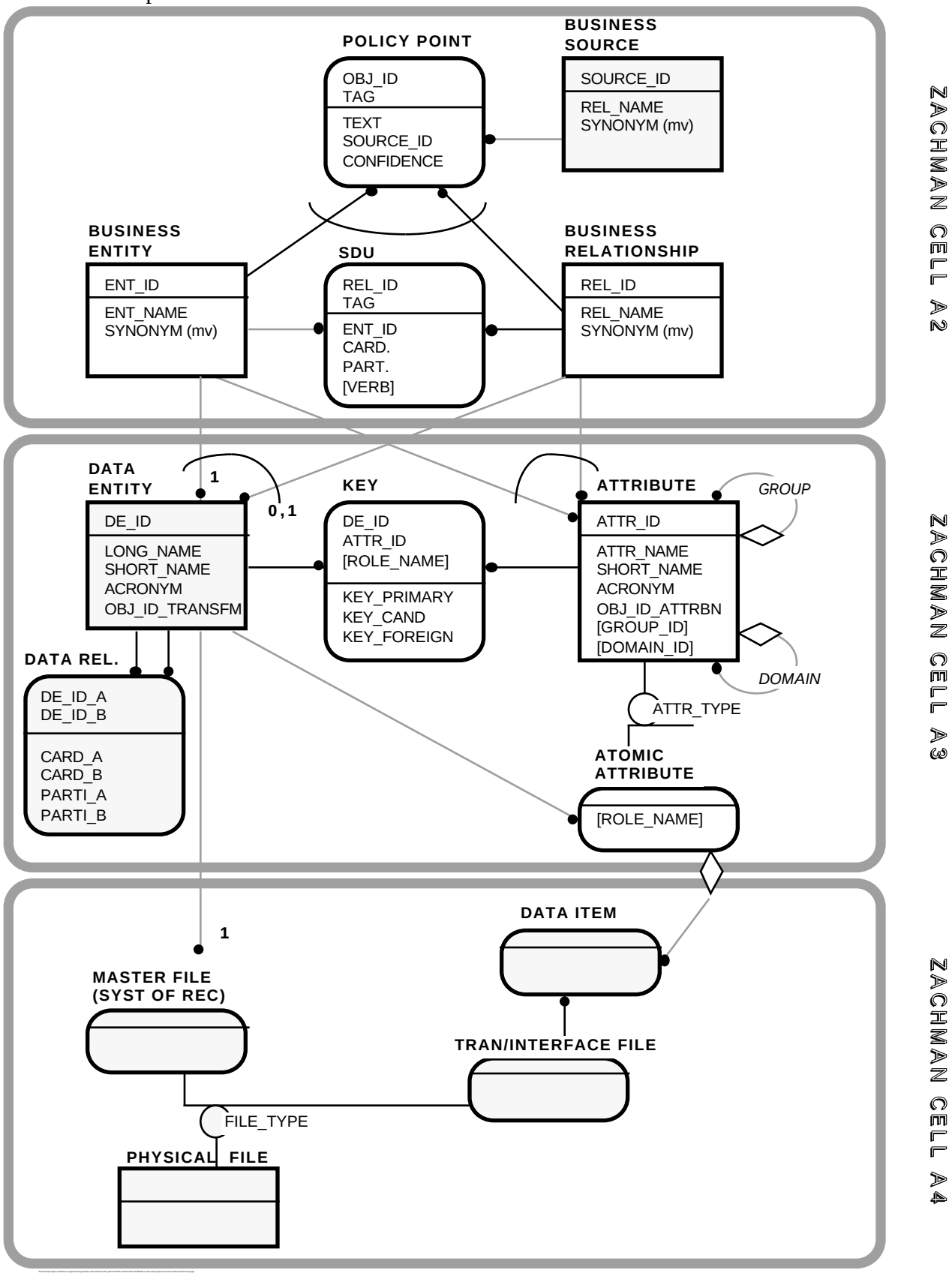
What better challenge for the data modeler than to use the concepts to model the concepts! This section presents a graphic *meta-model* of the concepts presented in the previous sections. The model is presented in both ER/IM and in DMT form. The translation into a Technology model in the form of relational tables is left as a *graduation exercise* for the reader.

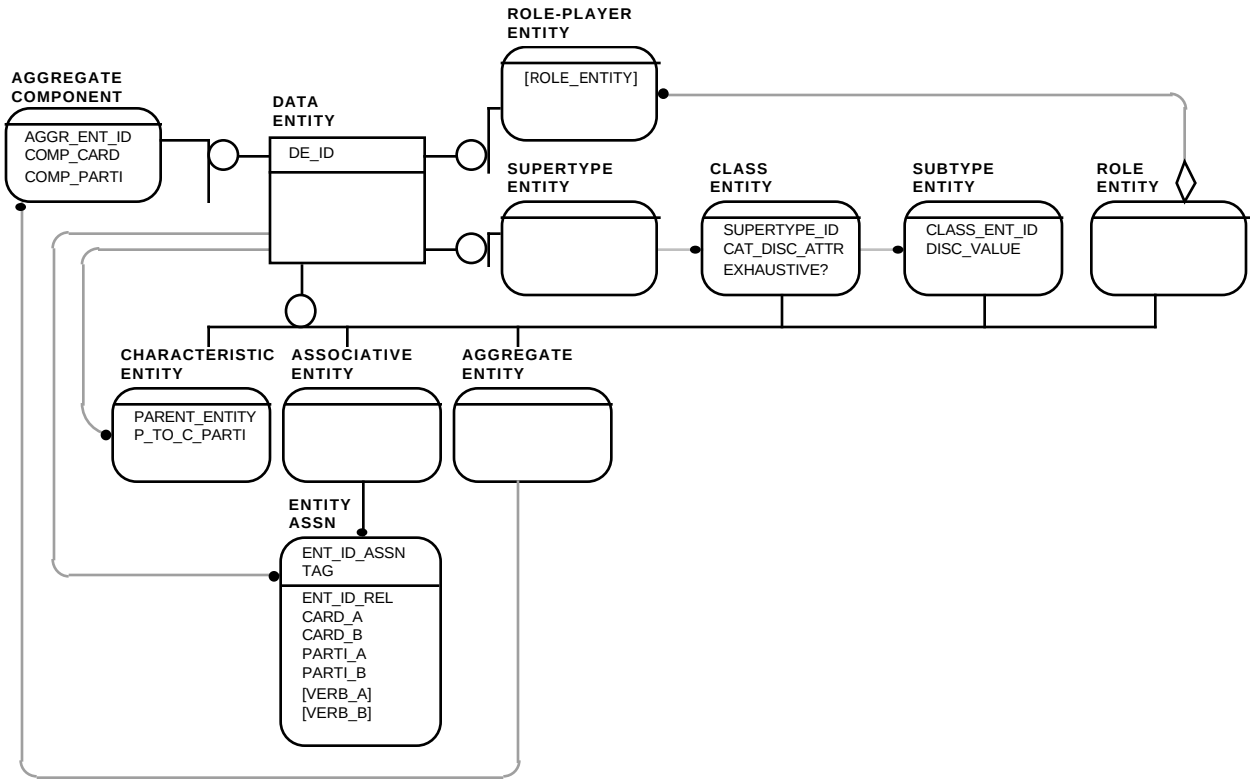
7.1 ER / IM PERSPECTIVE



7.2 I / S PERSPECTIVE

Below is a partial transformation of the Information Model in Sec. 7.1 into a CDM.




ROLE-PLAYER ENTITY
[ROLE_ENTITY]
AGGREGATE COMPONENT
AGGR_ENT_ID
COMP_CARD
COMP_PARTI
DATA ENTITY
DE_ID
SUPERTYPE ENTITY
CLASS ENTITY
SUPERTYPE_ID
CAT_DISC_ATTR
EXHAUSTIVE?
SUBTYPE ENTITY
CLASS_ENT_ID
DISC_VALUE
ROLE ENTITY
CHARACTERISTIC ENTITY
PARENT_ENTITY
P_TO_C_PARTI
ASSOCIATIVE ENTITY
AGGREGATE ENTITY
ENTITY ASSN
ENT_ID_ASSN
TAG
ENT_ID_REL
CARD_A
CARD_B
PARTI_A
PARTI_B
[VERB_A]
[VERB_B]

APPENDIX A – BNF FOR META-MODEL REPRESENTATION

Backus-Naur Form (BNF) is a shorthand for describing syntax rules. Its basic symbol,

::=

can be read as *"is defined to be."* Each clause begins with a keyword which identifies a specific clause type. The highlighted portion of the keyword represents an acceptable abbreviated form.

Other BNF symbols have standardized, specific interpretation as follows:

< …>	an instance value; this value is either the literal shown (upper-case) or a value that conforms to the domain rules of the name between the symbols (lower-case)
;	a delimiter between terms
[…]	an optional term or clause type
{ …|… }	"exclusive or" of terms between the braces; if there is a default choice, it is underlined
*	a clause type where there can be multiples within a clause set instance; otherwise there can be at most only one clause type for any instance of a clause set

The advantage of using a BNF definition is its independence from any existing product syntax, allowing it to reflect as rich a set of semantics as has been defined for the data model representation. It is relatively straight forward to implement a parser for a grammar that has been defined in this manner. In this fashion, the model can be translated into specific syntax for a variety of products.

The BNF clause sets and clause types defined to support the meta-model semantics described in this document are recapped below:

ENTITY

ENTITY::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>;<id_reference>; <text>
* [**CO**MMENT::=<tag[?]>; <id_reference>; <text>]

RELATIONSHIP

RELATIONSHIP::=<name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>; <text>
* [**CO**MMENT::=<tag[?]>; <id_reference>; <text>]

ASSOCIATION:

* **SD**U::=<tag>;<id_entitya>;<id_entityb>;[{M|1|n}] [{R|O|C|F }][: {M|1|n}] [{R|O|C|F }];
<verba>[;<verbb>]

GENERALIZATION:

GENERALIZATION::=<id_generalization_entity>;<id_attribute>[;<id_class_entity>]
;<valueset> [; { L |E}] [;{S|D}]

* **CA**TEGORY::=<id_category_entity>;<value>

ROLE-PLAYING:

ROLE**P**LAYING::=<id_role_entity>

* **RO**LESOURCE::=<id_source_entity>

AGGREGATION:

AGGREGATE::= <id_aggregate_entity> [; {S [B|D]|C}]

* **PA**RTOF::=<id_component_entity> ; [{1|M|n}] [{R|O|C}] [: {1|M|n}] [{R|O|C}]

CHARACTERISTIC:

CHARACTERISTIC::=<id_parent_entity>;<id_characteristic_entity> [;1: {M|1|n}] [{R|O|C}]

ATTRIBUTE

ATTRIBUTE::=<name>
IDENTIFIER::=<id>
* **SH**ORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::=<tag[?]>; <id_reference>;<text>]
DOMAIN::=<id_domain>
[**RO**LE::=<name>[;<name>...]]
PROPERTY::= {<id_entity>|<id_relationship>} [;P|A]
[**PA**RTOF::=<id_attribute>]

DOMAIN

DOMAIN::=<name>
IDENTIFIER::=<id>
DOMAIN**T**YPE::=<code>
* **IN**PUTAS::={Annn|Nnn.nn|D[J|MDY|DMY|YMD|C] [/|-]|T|LT}
* **ST**OREDAS::={Annn|Nnn.nn|D[J|MDY|DMY|YMD|C] [/|-]|T|LT}
* **VI**EWEDAS::={Annn|Nnn.nn|D[J|MDY|DMY|YMD|C] [/|-]|T|LT}
* [**V**ALUE**R**ULE::=<rule>]
* [**V**ALUE**P**ROC::=<procedure>]

VIEW

VIE**W**::= <name>
IDENTIFIER::=<id>
SHORTNAME::=<shortname>
* [**SY**NONYM::=<name>]
* **DE**FINITION::=<tag[?]>; <id_reference>;<text>
* [**CO**MMENT::= <tag[?]>, <id_reference>,<text>]
LEVEL::={ERM|CDM|DMB}
* [**V**IEW**O**F::=<id_view>]
* **V**IEW**A**TTRIBUTE::=<id_attribute>
* [**A**TTRIBUTE**C**OMMENT::= <tag[?]>; <id_attribute>;<id_reference>; <text>]
* **V**IEW**E**NTITY::=<id_entity>
* [**E**NTITY**C**OMMENT::= <tag[?]>; <id_entity>;<id_reference>; <text>]
* **V**IEW**R**ELATIONSHIP::=<id_relationship>
* [**R**ELATIONSHIP**C**OMMENT::= <tag[?]>; <id_relationship>; <id_reference>; <text>]
* **PR**OJECTID::=<id_project>
* **FU**NCTIONID::=<id_function>
PERSON**R**ESP::=<id_person>
* **P**ERSON**US**ER::=<id_person>

For each entity type, for change management and version control:

DATE**C**REATED::= <date>
DATE**U**PDATED_LAST::= <date>
[**DA**TE**A**UTHORIZED::= <date>]
[**DA**TE**I**NSTALLED:= <date>]
[**STA**TUS::={D|V|P}]
VERSION::= nnn.nn
PERSON**C**REATING::= <id_person>
* [**P**ERSON**UP**DATING::= <id_person>]
* [**P**ERSON**A**UTHORIZING::= <id_person>]

* Multiples allowed

BIBLIOGRAPHY

[ANDE89] Anderson Healy, Keri and Michael Eulenberg. "Data Modeling: Some Common Ground," The Data Base Newsletter, 17:2 (March/April 1989).

[BROW82] Brown, Robert G. Logical Database Design Techniques. The Database Design Group, Mountain View, Calif. (1982).

[BROW83] Brown, Robert G. ADAM – General Information Manual. The Database Design Group, Mountain View, Calif. (1983).

[CHEN76] Chen, Peter P-S. "The Entity-Relationship Model – Toward a Unified View of Data." ACM Transactions on Database Systems, 1 (March 1976):p. 9-36.

[CHEN83] Chen, Peter P-S. "English Sentence Structure and Entity-Relationship Diagrams." Proceedings of the E/R Conference. Anaheim, Calif. (Oct. 1983).

[CURT83] Curtice, Bob. "An Introduction to the Concept of Domains." Newsletter on Data Management. Arthur D. Little, Inc. (May, 1983):p. 5-26.

[DATE81] Date, Chris. An Introduction to Database Systems, 1, 3rd ed. Addison-Wesley Publishing Company (1981):574 pp.

[DATE86] Date, Chris. An Introduction to Database Systems, 1, 4th ed. Addison-Wesley Publishing Company (1986):639 pp.

[FLAV81] Flavin, Matt. Fundamental Concepts of Information Modeling. Yourdon Press (1981):128 pp.

[GILM87] Gilmore, Paul C. Database Design Using a Purely Set-based Conceptual Model. Presented at CIPS Data Administration SIG Meeting (January 26, 1987).

[GUID86] Repository Data Model Strategy Paper. GUIDE Publication No. GRP-153 (1986).

[GUID88] BSPI Methodology. GUIDE Publication No. [pending] (1988).

[KAMP86] Kampen, Garry. Conceptual Modelling. Seattle University, Seattle, WA. (1986).

[KENT78] Kent, William. Data and Reality – Basic Assumptions in Data Processing Reconsidered. North-Holland (1978).

[LEAR87] Learmonth, Roger. GUIDE talk (March, 1987): [get ref.]

[LOOM87] Loomis, Mary E. S. The Database Book. Macmillan Publishing Company, New York (1987):465 pp.

[POLL87] Pollard, David. Domains. Presented at Seattle DAMA Meeting (Sept. 8, 1987).

[ROSS87] Ross, Ronald G. Entity Modeling: Techniques and Application. Database Research Group, Inc., Boston, Mass. (1987):218 pp.

[SCHU87] Schuldt, Gary. Extending the E-R Model to include ROLES. [summary of presentation at the Information Resource Forum on 4 May, 1987] ©Syncretics.

[SMIT77] Smith, J. M. and D. C. P. Smith. "Database Abstractions: Aggregation." Communications ACM, 20 (June 1977):405-413.

[SMIT77B] Smith, J. M. and D. C. P. Smith. "Database Abstractions: Aggregation and Generalization." ACM Transactions on Database Systems, 2 (June1977):105-133.

[SOWA84] Sowa, John F. Conceptual Structures. Addison-Wesley, Menlo Park, CA (1984):481 pp.

[TASK87] Tasker, Dan. An Entity/Relationship View of Time. presented at the E/R Conference. New York, NY (Nov. 1987).

[TASK89] Tasker, Dan. 4th Generation Data. Prentice Hall, Sydney, Australia (1989).

[TSIC82] Tsichritzis, Dionysios C. and Frederick H. Lochovsky. Data Models. Prentice-Hall, Englewood Cliffs, NJ (1982):381 pp.

[ZACH87] Zachman, John. "A Framework for Information Systems Architecture." IBM Systems Journal, 26, No. 3 (1987):276-292.